To appear in ACM SIGGRAPH 97 Annual Conference Proceedings

Interactive Update Of Global Illumination Using A Line-Space Hierarchy

George Drettakis and François X. Sillion

iMAGIS[†]- GRAVIR/IMAG - INRIA

Abstract

Interactively manipulating the geometry of complex, globally illuminated scenes has to date proven an elusive goal. Previous attempts have failed to provide interactive updates of global illumination and have not been able to offer well-adapted algorithms controlling the frame rate. The need for such interactive updates of global illumination is becoming increasingly important as the field of application of radiosity algorithms widens. To address this need, we present a novel algorithm which provides interactive update rates of global illumination for complex scenes with moving objects. In the context of clustering for hierarchical radiosity, we introduce the idea of an implicit *line-space hierarchy*. This hierarchy is realized by augmenting the links between hierarchical elements (clusters or surfaces) with *shafts*, representing the set of lines passing through the two linked elements. We show how line-space traversal allows rapid identification of modified links, and simultaneous cleanup of subdivision no longer required after a geometry move. The traversal of line-space also limits the amount of work required to update and solve the new hierarchical system after a move, by identifying the modified paths in the scene hierarchy. The implementation of our new algorithm allows interactive updates of illumination after object motion for scenes containing several thousand polygons, including global illumination effects. Finally, the line-space hierarchy traversal provides a natural control mechanism allowing the regulation of the tradeoff between image quality and frame rate.

Keywords: Global illumination, Dynamic environments, Hierarchical radiosity, Form-factors, Interactivity, Frame-rate control.

1 Introduction

The use of realistic global illumination is becoming more and more widespread. As a consequence, users demand more flexibility, and better interaction with lighting systems. Ideally, a user would like to be able to interact with a scene and interactively perceive at least some degree of global illumination effects. A major limitation of current systems is the inability to move or change geometry in a scene, with simultaneous update of global illumination effects. Applications such as virtual studios, tele-conferencing in virtual environments, driving simulators etc., all require interactive manipulation of the scene geometry, without loss of important illumination information.

Although significant advances have been made towards accelerating radiosity calculations for changing geometry [7, 4, 11], all previous approaches fail to provide interactive global illumination updates even for moderately complex scenes, and do not provide a way to control the quality/speed tradeoff for interactive display. The new solution we present uses the subdivision of line-space implied by the link structure of hierarchical radiosity to achieve efficient interactive radiosity updates for scenes of moderate complexity.

The goal of our approach is to provide a unified framework which will allow a user of a hierarchical radiosity system to move objects in a lighting simulation, and interactively perceive global illumination changes. We also provide a mechanism permitting the user to sacrifice quality for speed, but still maintain at least some of the important visual cues due to global illumination.

In any hierarchical radiosity system, and in particular a clustering-based approach, the elements of the scenes are linked together following the potential interactions of light between such pairs. These links induce a subdivision of the line-space of the scene, or more accurately the space of line-segments, following the flow of light between scene elements. We augment links with an explicit representation of all lines passing between two elements via a *shaft* [9]. In particular, when an object moves, we can efficiently identify the parts of the system which are modified, by hierarchically descending in this line-space. This traversal permits efficient cleanup of the mesh where subdivision is no longer needed because of geometry changes, and allows us to mark the paths in the hierarchy which are modified. As a consequence, fast resolution of the modified part of the system of equations is achieved. Finally, the line-space hierarchy provides a natural way to control the expense incurred at every frame. This is achieved by limiting the descent into line-space by the time available at each frame.

We present an implementation of these ideas which shows that, using the line-space hierarchy we can achieve interactive updates of illumination (2-3 frames per second) for scenes of moderate complexity (up to about 14,500 input polygons). This includes the treatment of scenes almost exclusively lit by secondary illumination.

2 Context and Previous Work

The fact that the movement of an object often causes limited changes to a global illumination solution became evident early on in graphics research, in particular for "radiosity" algorithms. Several algorithms have been proposed which deal specifically with changes to geometry, and their evolution follows closely the progress of radiosity solutions. In what follows we present a brief overview of these methods.

2.1 Progressive radiosity solutions

The initial "full-matrix" radiosity solution [8], led to the development of an algorithm which took advantage of coherence properties of the hemi-cube, used to calculate form-factors [3]. This solution suffered from all the limitations of full-matrix radiosity (including

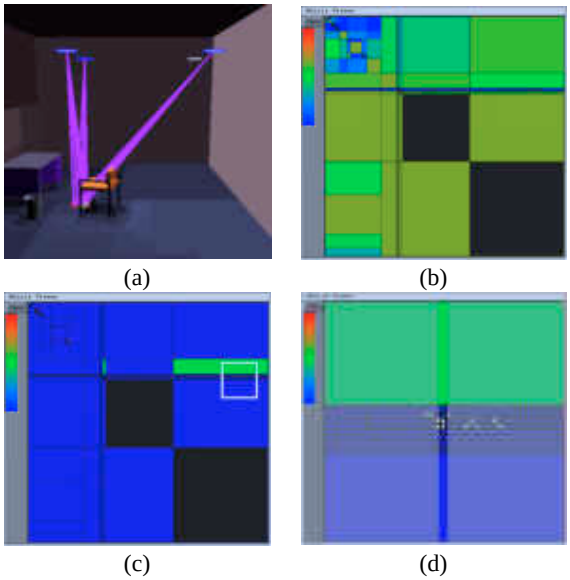
[†]iMAGIS is a joint research project of CNRS/INRIA/UJF/INPG. Address: iMAGIS/GRAVIR, BP 53, F-38041 Grenoble Cedex 09 France. Email: {George.Drettakis|Francois.Sillion}@imag.fr

quadratic storage and lack of adaptive subdivision), and most notably was limited to predetermined trajectories. Despite these drawbacks, this algorithm is noteworthy in the insight that form-factor changes can be limited spatially, by means of a swept volume restricting the part of space affected by the move.

The advent of progressive refinement solutions led to the development of two similar approaches which took advantage of the "shooting" process of radiosity propagation [4, 7]. By treating all object movements as deletions and re-insertions in the scene, shadows were removed by re-shooting energy, and new shadows were correctly inserted where appropriate by shooting "negative energy". This approach achieved impressive update times for direct illumination, even though global updates remained very expensive. An improvement to these methods was developed by Müller et al. [11], who added an intelligent data structure maintaining shadow-lists accelerating potentially modified interactions between surfaces. The main drawback of all these approaches is due to the fact that they are based on progressive refinement, for which the global energy balance is hard to control. In addition these methods cannot achieve interactive update rates (several frames per second) for scenes of moderate size or larger.

2.2 Hierarchical radiosity solutions

Some of the limitations of progressive refinement solutions can be addressed in the context of hierarchical radiosity. An overall "snapshot" of the global energy balance is maintained in the link structure, and the corresponding hierarchy. When an object moves, a limited number of links, and thus a limited part of the hierarchy, are modified. This can be seens by examining the block form-factor matrices resulting from hierarchical subdivision. In Fig. 1(a) the chair has just moved to the right. The block form-factor matrix (collapsed up to a certain hierarchical level for display) is shown in Fig. 1(b), "warmer" colors representing larger values of form-factors. The matrix in Fig. 1(c) represents the difference in the matrices produced by the move. As we can see, few regions of the matrix change.



(a) (b) (c) (d)

Figure 1: The chair in (a) has moved to the right. The block form-factor matrix is shown in (b), color coded (see text), before motion. In (c) we show the difference (dark blue = no change); (d) zoom of the modified matrix area, yellow blocks are the links shown in (a).

By zooming into a modified region (Fig. 1(d)), we realize that the changes are due to the moving shadow of the chair. In particular the region of the matrix selected in Fig. 1(d) corresponds to the element shown in red in Fig. 1(a), and the yellow blocks are a collapsed representation of the links arriving at this element.

Two first solutions have been developed exploiting hierarchical radiosity for dynamic environments. Forsyth et al. [6] present the idea of "promoting" and "demoting" links based on a refinement criterion which moves links up and down in the hierarchy, depending on the position of the objects at each frame. This is similar to the idea of "ghost" links presented by Shaw [12]. A first approach to the problems incurred by changes in visibility was presented in Shaw's work, by introducing special "shadow" links connected to the source and containing blocker information. A final optimisation was presented by which a "motion volume" is built with the bounding planes of the two positions of the dynamic object; All links are then tested against this volume to determine if they may have changed or not.

2.3 Shortcomings of previous methods

The algorithms for hierarchical radiosity in dynamic environments described above achieve significant improvement over the progressive refinement approaches developed earlier. Nonetheless, interactive updates rates are not achievable using these methods, especially for complex scenes, and since they were developed using traditional hierarchical radiosity, the quadratic cost of initial linking presents an important obstacle to their practical use.

More importantly, all previous approaches lack a unified mechanism which can rapidly identify the part of the system modified and at the same time control the simulation quality/time tradeoff in a coherent manner. In what follows we show that the *line-space hierarchy*, exploited by accessing the links attached to the scene hierarchy, provides this functionality. We will show how the hierarchical traversal of line-space allows rapid identification of the parts of the system which change, provides an efficient mechanism to update the hierarchy, and finally allows fine control of the quality/speed tradeoff for dynamic environments.

2.4 Objectives: user interaction with the scene

We assume that any object in the scene can be chosen to be dynamic. The only restriction is that each potential dynamic object must be included in a cluster of its own. When the user selects a dynamic object, the corresponding cluster is attached to the root of the hierarchy.

Changing the dynamic object during the simulation is not too complicated, since we can identify the links which were affected by the previous and the new dynamic object. This can be performed efficiently by traversing line space as described later, updating the links and the corresponding visibility information. Since the dynamic object clusters are attached to the root, a change in the object chosen can be accompanied by a re-insertion into the cluster hierarchy.

Once the object is chosen, the user can freely interact with the object to change its position (for example by translation or rotation). At each frame, all that is required is the previous and current position of the dynamic object bounding box. What is required next is to identify the links of the system whose shaft cuts either of these bounding boxes.

3 Hierarchical Line-Space Traversal

To rapidly identify the necessary modifications of the global illumination system of equations, we need a data structure which will isolate the parts of space which are affected by the motion of an object. The *line-space hierarchy* we introduce is such a representation encoded using the traditional link structure of hierarchical radiosity.

3.1 Introduction to the Line-Space Hierarchy

In the original hierarchical radiosity algorithm [10], the scene, as well as subsequent subdivision, is represented as a hierarchy. In addition, when the clustering algorithm is used, the polygons of the scene are grouped together to form a complete hierarchical representation of the environment. We will refer to such an element as an H-element h_e (*Hierarchical* element [13]), be it a cluster or a surface element. This scene hierarchy is augmented by *link* information, which is used to represent radiant exchanges between two H-elements of this hierarchy. A typical hierarchical solution process proceeds by *refining* the links: when the link is considered to insufficiently or incorrectly represent the light transfer, the element is subdivided and sub-links are created [10, 15]. The decision to subdivide is based on a "subdivision criterion" which may be based on error-estimation or magnitude of energy transfer across a link. We will refer to a "refiner", which is the module responsible for the refinement operation.

We represent the set of light paths in a hierarchical manner by augmenting the link structure. The shaft shown for example in Fig. 2(a) represents the entire set of lines which pass between the two elements. More precisely, this is the set of *line segments* rather than infinite lines. For simplicity however, we will use the term *line-space hierarchy* throughout this paper. In this respect, we build a coarse approximation to structures which encode visibility information of maximal free segments such as the visibility complex [5].

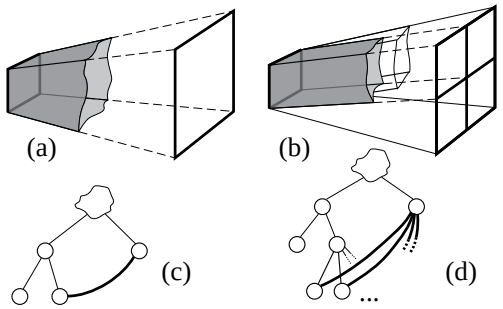


Figure 2: We show here a line space hierarchy: (a) the original link l^p (shown as a shaft) is subdivided, resulting in four sub-links (b), which are the line-space children of l^p. P-links (c) and the subsequent children links (d) are embedded in the H-element hierarchy.

When a link is subdivided, the four resulting links can be considered as children of the original link, Fig. 2(b). To store this hierarchy, subdivided links are not discarded, and are stored as *passive links* or *p-links* with the H-elements. Thus our new link hierarchy is actually *embedded* in the H-element hierarchy itself. For example, the thick black line in Fig. 2(c) corresponds to a (subdivided) p-link, and the four thick lines in Fig. 2(d) to the four resulting links. These ideas are related to the approach developed by Teller and Hanrahan [17], where a similar link hierarchy was used to incrementally maintain blocker lists, as well as the "ghost-link" idea [12] or link "demotion/promotion" approach [6].

3.2 Data structures

To explicitly work in line-space, we need a representation of the set of lines between surfaces. One possibility would be the approach using Plücker coordinates as was presented by Teller [16], or the intercepts of lines on two parallel planes [1]. Another approach would be that of ray-classification [2].

We have preferred to use the *shaft* structure [9] for its simplicity, and because it is well defined and easy to manipulate for the case when one endpoint of a link is a bounding box. In addition the shaft structure permits efficient intersection tests with bounding boxes, which is an operation central to the algorithms presented below. Since our algorithm operates in the context of clustering radiosity, this allows the use of the same structure for links between any combination of cluster and surface element. Another interesting aspect of the shaft representation is that shafts are truncated, and thus operate on line *segments*, as opposed to infinite lines.

The line-space hierarchy is built during the traditional refinement process of clustering hierarchical radiosity. The list of links arriving at a H-element is stored on the element itself, as well as all the p-links (see Figs. 2 and 3) Both active links and p-links are augmented with the shaft corresponding to the part of line-space they respectively cover.

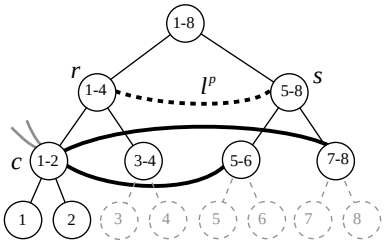


Figure 3: Numbering of the H-element hierarchy into index ranges.

One final component is required to allow efficient traversal the line-space hierarchy. Consider a given passive link $l^p_{s \rightarrow r}$, stored at H-element r and emanating from H-element s. Since this p-link has been subdivided, it is possible that the element s has been subdivided also. In addition, what originally was the p-link $s \rightarrow r$ is now a set of links at possibly different levels of the hierarchy, emanating from children of s, also at different levels. Links originating from other sources may also co-exist on the element r and its children. To be able to quickly identify the links which are children of $l^p_{s \rightarrow r}$, we need to be able to tell if a given link originates from an H-element which is a child of s. To avoid repeated traversals of the hierarchy which become overwhelmingly expensive, we apply a numbering scheme of all hierarchical elements. In particular, we traverse the entire hierarchy once at the beginning of the lighting simulation process, and assign an interval to each node, corresponding to the largest possible number of children which can be created. This number can be calculated based on the value of the *area threshold* A_ϵ [10], which limits the size of the smallest surface. To see this, consider the partially subdivided hierarchy shown in Fig. 3, where subdivided nodes are represented with solid lines. The index ranges assigned to each node correspond to the maximum subdivision which can possibly be incurred, shown as the complete tree, where currently unsubdivided nodes are shown with dashed lines.

3.3 Traversing Line-Space

Consider that we are situated on a given H-element h_e of the hierarchy and we wish to traverse the part of line space related to a p-link arriving at h_e. To effect the line-space descent, we need to visit exclusively the children links of the passive link considered. Each passive link l^p is connected to a source node s, which has a corresponding index range following the numbering scheme elaborated above. To visit the line-space children of l^p, we descend to each H-element child c of the current node h_e, and examine only those passive links originating from s or s's children, arriving at c. These are the links attached to a node with an index within the interval of the index of s.

For example, in Fig. 3, we are at receiver node r. P-link l^p is the thick dashed line. When descending to its line-space children, we visit for example H-element c. The only links we consider however are those emanating from H-element in the range $[5-8]$, i.e., children of the original H-element s, linked by l^p. This procedure is summarized in Fig. 4.

```
traverseLineSpace(H-element he, IndexRange ind)
{
    for each p-link l^p of he
        // find the "source" node q
        H-element q = l^p →LinkedNode()
        if( ind contains q→IndexRange() )
            for each child c of he
                // limit traversal to the index of q
                traverseLineSpace( c, q→Index )
}
```

Figure 4: Hierarchical Line Space Traversal

3.4 Efficient illumination update

To perform an update we first need to find the links affected by object motion. We consider as potentially changed the links whose *shafts* are touched by the bounding box of the *dynamic object* (i.e., the object which moves) before or after the move. After motion, the subdivision of parts of the hierarchy (for example due to shadow motion) may no longer be needed. We need to identify these parts of the hierarchy and perform the cleanup.

For the links which have been thus identified, new form-factor values need to be computed, since visibility may have changed with respect to the dynamic object. We call this operation *link update*.

To perform form-factor update efficiently, we maintain two-part occlusion information with each link: occlusion with respect to the dynamic object, and occlusion with respect to the rest of the scene. Thus at each frame, visibility is checked only with respect to the dynamic object. This information must of course be updated when we change dynamic objects; a line traversal to find the sub-spaces affected by the old and new object suffices to achieve this. Also, when a new link is created, its visibility with the rest of the scene is computed (but not for reinstated p-links).

After link update, it may be the case that certain links have to be refined, in particular when a visibility change occurs. The new values of the radiosity need to be computed and gathered on the links. The hierarchical system must then be updated to reflect the new position of the moving object. Finally the new system must be solved. The line-space hierarchy provides the necessary mechanism to efficiently perform all of the above steps. We thus summarize our new approach as follows:

1. Find the links potentially changed by traversing line-space and remove subdivision unnecessary due to geometry change;
2. Update the modified links and refine where necessary;
3. Solve the hierarchical system efficiently.

Step (1) occurs during the line-space traversal, resulting in the information necessary to perform steps (2) and (3).

4 Rapid Identification of Modified Links and Subdivision Cleanup

We next show how line-space traversal can be used to rapidly identify the links which have potentially changed. This traversal allows simultaneous removal of subdivision which is no longer required due to the new position of the moving object.

4.1 Finding the modified links efficiently

To find the modified links, the line-space traversal can be thought of as "zooming-in" to the region of space which has changed. The traversal algorithm starts at the root of the hierarchical elements (a cluster whose extent is the bounding volume of the scene), and descends recursively. At a given hierarchical node h_e, we visit all its passive links. For each such p-link l^p, we determine whether its corresponding line-space (represented by the shaft), was affected by the object motion. This is performed by testing the link shaft against the previous and current positions of the dynamic object. If the link was affected, we will potentially descend into its corresponding line-space. Given the new position of the dynamic object, we first test whether passive link l^p would no longer satisfy the subdivision criteria; this could occur for example if the p-link was partially occluded by the previous position of the dynamic object, but is completely unoccluded in the new position. If this is the case, the link can be reinstated as active, and the descent into the line-space hierarchy is stopped. If, on the other hand, the passive link l^p is maintained, the line-space children of l^p are visited, and the same process is applied recursively.

Finally, when the traversal of the passive links of h_e is complete, we examine all of the active links corresponding to the current part of line space (links originating from s or s's children) to determine if they are affected by the object motion. If this is the case, i.e., the active link shaft cuts the dynamic object before or after the move, it is added to a list of candidates for update and potential refinement. An example of a candidate list is shown in Fig. 5(b); notice that few of the numerous links describing energy exchanges in the left room or with the left wall of the right-hand office appear in this list.

4.2 Cleanup of unnecessary subdivision

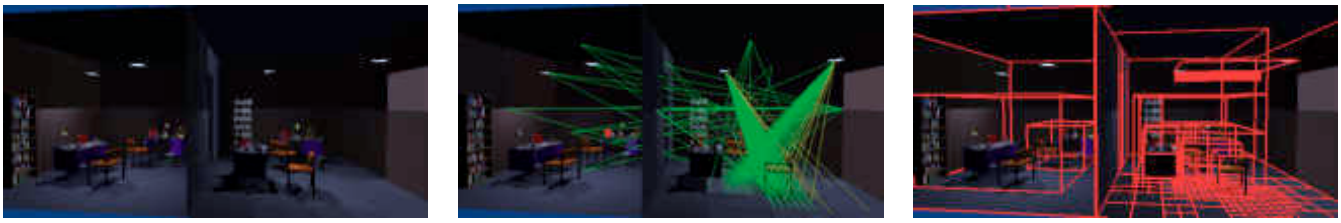
The traversal described above also provides the benefit of cleaning up unneeded refinement in the same step as the determination of the links which need to be updated. In particular when we have decided that a p-link l^p (linked to H-element s) on H-element h_e does not merit refinement due to the new position of the dynamic object, we can remove the entire set of links and p-links which are (line-space) children of l^p. This is performed by descending to the children of H-element h_e and removing all passive and active links linked to s and its children. After this removal, if there are no links remaining on any of the children of h_e (or its intermediate nodes), the subdivision is cleaned up. A similar approach to subdivision cleanup presented in [12] required a special pass, and multiple hierarchy passes to mark attached sources and their children.

The line space traversal is summarized in Fig. 6. In this algorithm, notice that when a potential change is found (either for a p-link re-installation or the required update of an link), the corresponding H-element is marked changed. For example, the H-elements marked "changed" are shown in red in Fig. 5(c). This will be used in what follows to limit the system solution at each frame. In addition, this approach is particularly efficient in the context of a clustering algorithm, since the number of links is limited.

5 Fast System Solution By Hierarchy Pruning

The line-space traversal algorithm described above results in fast identification of the links which need to be refined and at the same time allows the un-refinement of unnecessary subdivision. Once this step has been performed, we need to update the links which are changed, and potentially refine certain links. This additional refinement will be required for example around new shadow boundaries. Once these steps are complete, we have a complete hierarchical system which is ready to be resolved. In hierarchical radiosity [10, 15], system resolution is performed by performing complete sweeps of the hierarchy: *Gather* and *Push-Pull*. In the context of clustering [13], irradiance is gathered through the links onto the H-elements; this irradiance is subsequently pushed down the hierarchy. Finally, the radiosity values are computed at the leaves, and pulled up the hierarchy to provide the new solution.

Such an approach was used for a dynamic solution in [12] for example. For large scenes containing thousands of input polygons,



(a) (b) (c)

Figure 5: This example contains 14,572 input polygons. The chair in the right-hand room moves to the right (a) - (b). The average update time is approximately .3 s. In (b) we show the links modified and in (c) the parts of the hierarchy affected by the move and marked "changed" by the line-space traversal. Note how the several thousand surfaces of the left-hand room are unaffected.

```
findModifiedLinks(H-element he, IndexRange ind)
{
    for each p-link pl of he {
        H-element q = pl→LinkedNode()  // "source" node
        if( ind contains q→IndexRange()
            and affectedByMotion(pl→Shaft()) {
            if( wouldBeRefined( pl ) ) {
                for each child c of he
                    // limit traversal to the index of q
                    findModifiedLinks( c, q→IndexRange() )
                    if a child changed
                        he→setChanged()
            }
            else
                reInstateLink( pl )
                he→setChanged()
        }
    }
    // if any links of he are modified they are added
    // to the list of links to refine, and he is marked changed
    checkAddLinks( he )
}
```

Figure 6: Efficient Modified Link Identification

these multiple sweeps of the hierarchy result in an unacceptable overhead in an interactive context. Furthermore, such complete hierarchy traversal is unnecessary: only a small number of elements has actually been changed.

In what follows, we show that we can perform a *Gather* exclusively for links which have been affected by object motion, during link update. The line-traversal algorithm returns a list of links potentially changed. The form-factors for these links are recomputed to reflect the new position of the dynamic object. This update operation may result in the need for certain links to be refined.

Once the refinement is complete, the global solution can be efficiently performed. In particular the marking of the changed paths performed during line-space traversal allows us to limit the *Push-Pull* to the subsections of the hierarchy which have been modified, for one bounce of illumination. Subsequent bounces may potentially be required but can also be treated efficiently.

There is one essential assumption for the following discussion: we assume that before any motion is performed, the system has run to convergence. We define a solution as converged when no more links can be refined for the given error threshold, and the energy balance has been computed using all links.

5.1 Link update and in-place *Gather*

Once a link has been identified by the line-space traversal algorithm as potentially changed, we need to update its form-factor value. There are two possibilities: either the link has changed little and as a consequence it will not be refined or the link has changed in a way which requires further subdivision.

In the first case we need to modify the irradiance of the receiving patch by the difference of the previous irradiance and the current irradiance. For source s and receiver r, B_s the radiosity of H-element s, F^k the form-factor before the move and F^{k+1} the form-factor after the move, this difference is simply:

$$I_{diff} = B_s \left(F^{k+1} - F^k\right) \tag{1}$$

For clusters, we also need to distribute the irradiance down to the cluster contents.

When refinement occurs, we need to remove the irradiance which arrived at the receiving node from the previous transfer on the link. After subdivision, the new irradiance will be transferred when the sub-links are established. For element r, the irradiance I_r becomes:

$$I_r^{k+1} = I_r^k - F^k B_s \tag{2}$$

After subdivision, the new links will be established at a different level in the hierarchy, and we then simply add in the new irradiance. We thus avoid the *Gather* sweep of the hierarchy.

5.2 Non-recursive refinement for modified links

At each frame, certain links will be identified as requiring refinement. This typically occurs when the dynamic object touches a link shaft for the first time. Previous hierarchical radiosity (e.g., [10]) performed recursive refinement of links. In such an approach, when a link is refined, we immediately attempt to refine its children and so on recursively until subdivision is no longer possible given the current subdivision criteria.

For the requirements of controlling the solution, we need to achieve two goals: (a) refine the most important links first and (b) potentially truncate the refinement process if we wish to limit the amount of time spent for a frame. The latter point becomes essential for the time/quality control algorithm presented in Section 6.1.

To achieve this, when refining a link, the resulting refined sub-links are added to a heap. The refiner then extracts the link with the highest potential power transfer for refinement, thus achieving goal (a). The refiner keeps track of the number of links it still needs to refine n_l, and the number n_r of links already refined. When the sum of n_l and n_r exceed the limit fixed by the subdivision process, the refinement is terminated. The n_l form-factors of the remaining links are then updated to correspond to the new position of the dynamic object, and established at the given level without being refined.

5.3 Global solution

Once the line-space traversal and the refinement are complete, we have a new hierarchy for which the irradiance at each node corresponds to the new position of the object. In addition, we have marked as "changed" all the elements in the hierarchy which have been modified, as well as all the paths in the hierarchy leading to these elements. The only remaining step is the *Push-Pull* process of

the hierarchical solution which will result in the correct hierarchical representation of radiosity at every level of the hierarchy.

As mentioned earlier, we exploit the information of the paths and elements marked "changed", to accelerate the *Push-Pull* since it will only be performed on a small part of the hierarchy.

5.3.1 Single Bounce

We modify the *Push-Pull* procedure to visit only the parts of the hierarchy which are modified. At a given node, we check if it is marked "changed": if it is we proceed as normal, descending to the children, and if not we use the radiosity already calculated at this level instead of continuing the hierarchy descent. This procedure is summarized in the pseudo-code of Fig. 7.

```
Spectrum partialPushPull(Helement* he, Spectrum IrradDown)
{
    Spectrum RadUp
    if he→changed() { // normal PushPull
        if ( he→isLeaf() )
            RadUp = ComputeLeafRad
        else foreach child c of he
            RadUp += partialPushPull(c, IrradDown + he→Irrad())
    }
    else // use previous values
        RadUp = he→Radiosity()
    he→Radiosity = RadUp
    return RadUp
}
```

Figure 7: Partial Push-Pull

This update algorithm will result in a system which is updated to reflect the new position of the dynamic object. This new state includes changes to all links, for direct *and* indirect illumination. In most cases, this is largely sufficient, since it represents a system which in many cases has converged. This solution always provides updated shadows due to primary illumination, as well as for some shadows due to indirect illumination.

5.3.2 Subsequent bounces of illumination

It may be the case however that the system has changed sufficiently so that subsequent iterations are needed to achieve convergence. Consider the example of a dark room with a door (initially closed) opening to a bright corridor (see Fig. 10). When opening the door, some direct light will flow into the room due to links which will be refined. If the previous algorithm were to be used as described above, certain light transfers would not occur (e.g., from the floor to the ceiling), since the radiosity values in the hierarchy would not be modified until after the call to $partialPushPull$.

To determine the modified links, we perform a partial traversal of the hierarchy, visiting only the elements marked "changed" by the line-space traversal. At each H-element h_e, the H-elements it is linked to are stored, that is the H-elements for which h_e acts as a "source." In addition, we store the old value of radiosity, corresponding to the previous dynamic object position. Thus at every H-element changed by the first bounce, we can calculate the difference in irradiance, and perform an in-place gather to the receiving node as described above. Some links may need to be refined, and as such are inserted into the heap. Subsequent refinement is then performed, followed by a new partial push pull. Care must be taken to re-initialize the "changed" markings of every node, as well as the old value of radiosity after each iteration. In particular, H-elements updated by the difference in irradiance are marked as "changed" as well as their parents and children. The difference in global effects is generally limited, and thus the extents of these updates is not very large, even for cases where global illumination effects are very important (e.g., Fig. 10).

6 Controlling Simulation Time and Rendering Quality

The solution presented above allows rapid updates for scene of moderate complexity. Nonetheless, the update rate can be too slow for certain applications where 1 or 2 frames per second is simply not acceptable. In addition, for very complex scenes, we need to be able to provide the user with the choice of "give me N frames per second, with the best possible quality."

The limitation of the algorithm presented above is that there is no control on the number of links that need to be updated. Thus, if the dynamic object moves into a part of space which contains a large number of links, the line-space traversal will create a large candidate list. A large amount of time will thus be required to update the form-factors of these links, and to potentially refine some of them.

To avoid this problem we present an algorithm that updates as much of the modified link hierarchy as possible, in the sense of a user-defined time limit, and unrefines the rest. This can be performed naturally with the aid of the line-space hierarchy, and is very much in the spirit of the original hierarchical radiosity approach.

6.1 Controlled update algorithm

To achieve a controlled update time the user first selects a target frame rate. The system then calculates the number of link updates that it can perform in a given frame. Thus we consider as our "time" or cost unit, the time required for a link update.

To achieve the control of the number of link updates, and adapt the hierarchy to the desired frame rate, we first calculate the number of child links and p-links for each p-link in the hierarchy. This is performed while traversing line-space to find modified links, as presented above in Fig. 4, and "pulling" the number of links resulting from the subdivision of each p-link.

To perform the link update, we traverse line-space for a second time. During this sweep, we only traverse the parts of the hierarchy actually modified.

The update control algorithm is very similar to the line space traversal. The important difference is that we always update the links of the current node *first*. We are thus assured that the state of the hierarchy above the current node h_e is correctly up to date. This is an essential requirement, since we can not otherwise truncate the update without incurring inconsistencies. The time spent updating links of this node is subtracted from the remaining time.

We then compute the minimum amount of work required if we are to descend in line-space. This is equal to the total of all line-space children active and passive links arriving at the children of h_e. If this number is larger than the remaining "time", we cannot guarantee that we will be able to update the remaining sub-links in the allotted time. In this case, we reinstate the appropriate p-links of h_e and cleanup underlying links (and remove subdivision if necessary). We then stop the descent.

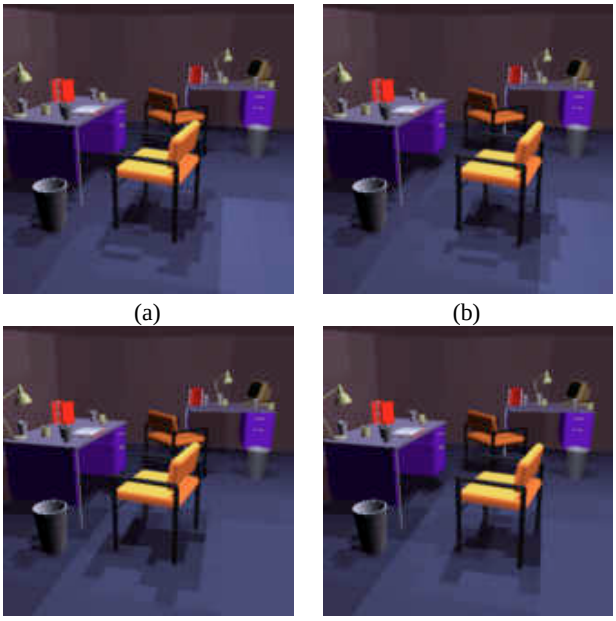
If we can still descend, we continue the traversal of line-space. We assign to each child the fraction of time calculated as follows. We sum the total number n_{cl} of children links stored with each p-link of h_e, in the current index range. The fraction assigned to each child c of h_e is the number of active sub links of p_l arriving at c plus the number of active sub links below c, also stored during the push-pull of line-space. This can be seen as a local and adaptive form of progressive multi-gridding [14].

A very important feature of this controlled update strategy is that no matter how much time is allocated for the update, a consistent solution is computed. This is due to the fact that the set of links always completely covers the entire line space. The availability of a *complete* hierarchy is a benefit of using hierarchical clusters. In other words, the algorithm ensures that all possible energy transfers are accounted for, although they may be included in links very high

```
controlledUpdate(Helement he, IndexRange ind, int remainingCost )
{
    updateLinks(h_e)
    remainingCost = remainingCost− links updated
    plinkCost = number of children p-links and links in ind range
    if pLinkCost ≤ remainingCost {
        for each p-link l^p of he
            H-element q = l^p  →LinkedNode()  // "source" node
            if ind contains q  →IndexRange()
                for each child c of he with c  →changed()
                    cost = fraction of child links of l^p in c
                    controlledUpdate( c, q  →IndexRange(), cost )
    }
    else
        update and reinstate all p-links of h_e in ind range
}
```

Figure 8: Controlled Update Algorithm

up the hierarchy (probably as a significant approximation, inducing some error). This consistency comes in contrast to all previous approaches to incremental updates of radiosity solutions, and is an important asset for many practical applications.



(a) (b) (c) (d)

Figure 9: The chair is the dynamic object, moving to the right, with no additional bounces. Update rates (a)-(b) average .5s. In (c) and (d) the same motion is shown with time/quality tradeoff, limited to 330 link updates per frame. The average update takes .3s.

6.2 Maintaining consistent quality

The controlled line-space descent presented above performs well for many configurations. Nonetheless, due to the quadtree nature of the subdivision on surfaces, it is often the case that new refinement does not occur as desired. In particular we may notice that subdivision is not propagated across quadtree edges.

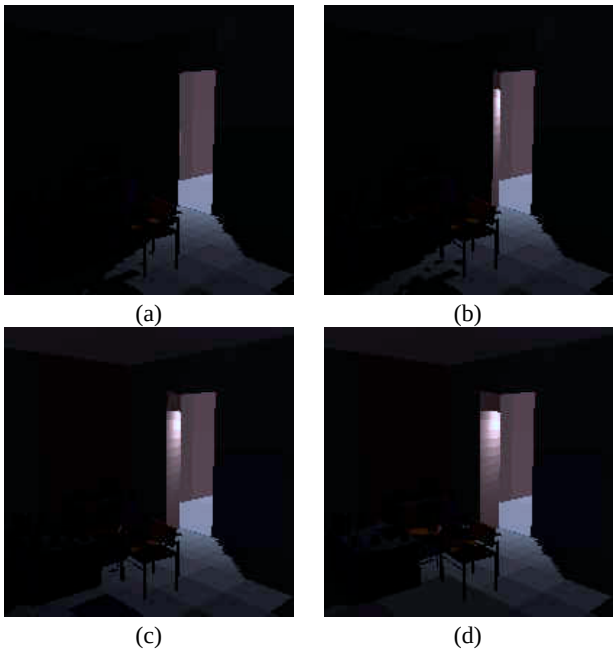
To counter this problem, we influence link refinement. Consider a p-link inserted into the heap for refinement because the dynamic object cuts its shaft, and that there was no interaction in the position at the previous frame. In this case, we increase the key used to sort elements in the heap, making refinement of this link more probable.

7 Implementation and Results

We have implemented the new algorithms on a clustering hierarchical radiosity algorithm following [13]. The modules for line-space traversal and all modified refinement routines required around 5000 additional lines of C++. All test results are reported on an Indigo 2 Impact, with an R4400 processor at 200MHz.

7.1 Basic algorithm

The first example is a single scene containing 5,405 polygons shown in Fig. 9, with four light sources. For this scene, we show one chair moving to the right, which is an object containing 870 polygons. The update rates are close to 0.5 seconds per frame, which can be satisfactory in many cases. The breakdown of the computation time for the first move for example is: 0.08 s. for line-space traversal, 0.40 s. for link update/refinement and 0.05 s. for partial push-pull.



(a) (b) (c) (d)

Figure 10: In this example, a door opens onto a dark room (5,295 polygons). Additional bounces are performed, and global illumination is updated. Updates takes (a)-(b): 3.20 s.; (b)-(c): 3.47 s.; (c)-(d): 2.52 s.

The second example is shown in Fig. 10, where the additional iterations are required. Initially the room is completely black. While the door is gradually opened, global illumination is rapidly updated. We see that a reasonable representation of global illumination is provided with update rates of around 3 seconds per frame. For the second update for example, the single bounce took a total of 2.15 seconds and the subsequent two bounces 0.55 and 0.5 seconds each.

The third example shows that we can handle complex geometry without significant degradation in speed. In Fig. 5, we have a scene with 14,572 input polygons, with the chair in the right-hand room moving to the right. The update takes around 0.6s. per frame, but sometimes improves to 0.2s per frame (5 fps) when the chair traverses "sparse" regions of line-space (i.e. with few links).

7.2 Time/quality tradeoff

The controlled update algorithm was tested on the scene used previously to demonstrate the basic approach. In Fig. 9(c) and (d) we present the output of the algorithm for a selected value of 330 link updates per frame, which results in an average update rate of approximately .3 seconds/frame. The shadows are of lower quality compared to the image shown in Fig. 9, but are still definitely usable.

7.3 Higher quality movement

Our approach is also well adapted if the user requires higher quality, and is prepared to wait longer (several seconds). An example is shown in Fig. 11, where an update takes about 4 s., but we see that the quality of shadows is improved compared to Fig. 9. To achieve this, we simply decrease the ϵ threshold for BF refinement [10].

8 Conclusions and Future Work

We have presented a new framework for the efficient calculation of incremental changes of global illumination using hierarchical radiosity. This set of algorithms allows interactive updates of the lighting solutions in scenes with moving geometry. The heart of our approach is the introduction of a *line-space* hierarchy associated with the links between scene elements. This hierarchy is distinct from, but related to, the hierarchy of surfaces and clusters in the scene, and can be traversed efficiently using the implicit correspondence between the two hierarchies. In terms of data structures, the main addition with respect to hierarchical radiosity is the introduction of shaft structures representing a portion of line segment space associated to each link.

The line-space traversal algorithm is beneficial in many respects. First, it allows the fast identification of the links that should be modified in response to object motion. These links are collected to accelerate later processing. In addition, it permits simultaneous cleanup of object subdivision that has become unnecessary due to visibility changes induced by object motion. The line-space traversal marks the modified parts of the hierarchy, resulting in an accelerated solution of the modified system.

Figure 11: Higher-quality solution, 4 s. per frame on average.

The set of links marked during line-space traversal embodies all of the form factor changes in the scene, although at a hierarchical level which may not be sufficiently detailed for the desired accuracy. Hierarchical refinement of these links is carried out in a non-recursive manner, using a set of refinement criteria to resolve fine radiosity variations and shadow details. The marking mechanism results in a very fast solution of the updated system of equations.

Finally, the line-space hierarchy also provides a natural mechanism to control the time/quality tradeoff for incremental updates, by constantly monitoring the expected cost of refinement operations and ensuring that the refinement budget is not exceeded. The nature of the refinement algorithm ensures that the solution is always consistent in that it takes into account all possible energy transfers in the scene, although possibly at high hierarchical levels.

Future work includes the design of improved quality control mechanisms. The very notion of solution quality is difficult to define. For instance, it is well accepted that the presence of shadows for moving objects is an important visual cue for understanding object location and motion, but these shadows need not necessarily be very precise. Recent work on accelerated approximate shadow calculations using the hierarchy of clusters and feature-based error metrics may be applicable to dynamic updates [14].

Finally, more involved visibility structures, which provide detailed line-space information, such as the Visibility Complex [5], will probably be very useful for dynamic updates of illumination.

Acknowledgements

Thanks to Frédo Durand, Mathieu Desbrun and Rachel Orti for crucial help.

References

[1] N. Amenta. Finding a line transversal of axial objects in three dimensions. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 66–71, 1992.

[2] James Arvo and David Kirk. Fast ray tracing by classification. *Computer Graphics*, 21(4):55–64, July 1987. Proceedings SIGGRAPH '87 in Anaheim.

[3] Daniel R. Baum, John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. The back-buffer algorithm : an extension of the radiosity method to dynamic environments. *The Visual Computer*, 2:298–306, 1986.

[4] Shenchang Eric Chen. Incremental radiosity: An extension of progressive radiosity to an interactive image synthesis system. *Computer Graphics*, 24(4):135–144, August 1990. Proceedings SIGGRAPH '90 in Dallas.

[5] Frédo Durand, George Drettakis, and Claude Puech. The 3d visibility complex, a new approach to the problems of accurate visibility. In X. Pueyo and P. Shroeder, editors, *Rendering Techniques '96*, pages 245–257. Springer Verlag, June 1996. Proc. 7th EG Workshop on Rendering in Porto.

[6] David Forsyth, Chien Yang, and Kim Teo. Efficient radiosity in dynamic environments. In Sakas et al., editor, *Photorealistic Rendering Techniques*, Darmstadt, D, June 1994. Springer Verlag. Proc. 5th EG Workshop on Rendering.

[7] David W. George, François Sillion, and Donald P. Greenberg. Radiosity redistribution for dynamic environments. *IEEE Computer Graphics and Applications*, 10(4), July 1990.

[8] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(3):213–222, July 1984. Proc. SIGGRAPH '84 in Minneapolis.

[9] Eric A. Haines. Shaft culling for efficient ray-traced radiosity. In Brunet and Jansen, editors, *Photorealistic Rendering in Comp. Graphics*, pages 122–138. Springer Verlag, 1993. Proc. 2nd EG Workshop on Rendering (Barcelona, 1991).

[10] Pat Hanrahan, David Saltzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics*, 25(4):197–206, August 1991. SIGGRAPH '91 Las Vegas.

[11] Stefan Müller and Frank Schöffel. Fast radiosity repropagation for interactive virtual environments using a shadow-form-factor-list. In Sakas et al., editor, *Photorealistic Rendering Techniques*, Darmstadt, D, June 1994. Springer Verlag. Proc. 5th EG Workshop on Rendering.

[12] Erin Shaw. Hierarchical radiosity for dynamic environments. Master's thesis, Cornell University, Ithaca, NY, August 1994.

[13] François Sillion. A unified hierarchical algorithm for global illumination with scattering volumes and object clusters. *IEEE Trans. on Vis. and Comp. Graphics*, 1(3), September 1995.

[14] François Sillion and George Drettakis. Feature-Based Control of Visibility Error: A Multiresolution Clustering Algorithm for Global Illumination. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 145–152. ACM SIGGRAPH, August 1995.

[15] Brian E. Smits, James R. Arvo, and David H. Salesin. An importance-driven radiosity algorithm. *Computer Graphics*, 26(4):273–282, July 1992. Proc. SIGGRAPH '92 in Chicago.

[16] Seth J. Teller. Computing the antipenumbra of an area light source. *Computer Graphics*, 26(4):139–148, July 1992. Proc. SIGGRAPH '92 in Chicago.

[17] Seth J. Teller and Patrick M. Hanrahan. Global visibility algorithms for illumination computations. In J. Kajiya, editor, *SIGGRAPH 93 Conf. Proc. (Anaheim)*, Annual Conf. Series, pages 239–246. ACM SIGGRAPH, August 1993.